Molecular Dynamics for Very Large Systems on Massively Parallel Computers: The MPSim Program

KIAN-TAT LIM,[1] SHARON BRUNETT,[2] MIHAIL IOTOV,[1] RICHARD B. McCLURG,[1] NAGARAJAN VAIDEHI,[1] SIDDHARTH DASGUPTA,[1] STEPHEN TAYLOR,[2] and WILLIAM A. GODDARD III[1]*

[1]Materials and Process Simulation Center, Beckman Institute (139-74), Division of Chemistry and Chemical Engineering and [2]Scalable Concurrent Programming Laboratory, California Institute of Technology, Pasadena, California 91125

Received 18 August 1995; accepted 12 June 1996

ABSTRACT

We describe the implementation of the cell multipole method (CMM) in a complete molecular dynamics (MD) simulation program (MPSim) for massively parallel supercomputers. Tests are made of how the program scales with size (linearly) and with number of CPUs (nearly linearly) in applications involving up to 10^7 particles and up to 500 CPUs. Applications include estimating the surface tension of Ar and calculating the structure of rhinovirus 14 without requiring icosahedral symmetry. © 1997 by John Wiley & Sons, Inc.

Introduction

Large-scale systems with millions of atoms are of great interest in many areas of chemistry, biochemistry, and materials science. Such systems include viruses such as rhinovirus (common cold) and poliovirus that contain nearly one million atoms[1]; starburst dendrimers[2] where self-limited growth of PAMAM may involve molecules with about 250,000 atoms; and commercial polymers where typical molecular weights of 10 million daltons would lead to chains with millions of atoms. Atomistic molecular dynamics (MD) simulations of such large systems are important to provide the atomic detail needed to specify chemical properties such as binding of a drug to a virus or mechanical properties such as the modulus.[3]

*Author to whom all correspondence should be addressed. E-mail wag@wag.caltech.edu

This article includes Supplementary Material available from the authors upon request or via the Internet at ftp.wiley.com/public/journals/jcc/suppmat/18/501 or http://www.wiley.com/jcc

In developing methodologies for routine applications of MD to million-atom systems, it is important to retain accuracy while ensuring that the computational costs scale linearly with size.[4, 5] The key problems here are the long-range Coulomb (Q) and van der Waals (vdW) interactions, collectively referred to as nonbond (NB) interactions. Since the NB interactions include every pair of atoms, exact calculations require $O(N^2)$ operations, where N is the number of atoms, which is not feasible for million-atom systems. The usual approach is to truncate the interactions at some cutoff distance. This reduces the operation count to $O(N)$, but with a significant decrease in accuracy, particularly for the Coulomb interactions.[6]

Fast multipole methods[7] were developed to overcome these limitations. By treating the long-range interactions as multipole expansions over successively larger regions, they avoid the inaccuracy of cutoffs while obtaining $O(N)$ scaling with size. They can be used for any inverse power-law interaction potential[4] including Coulomb $(1/R)$, screened Coulomb $(1/R^2)$, vdW dispersion $(1/R^6)$, and Pauli orthogonalization $(1/R^9$ or $1/R^{12})$. The cell multipole method (CMM)[4] is a particularly regular type of multipole method that is easily implemented on massively parallel computers. In addition, CMM has been extended to infinite systems while retaining the $O(N)$ scaling.[5] Previous tests[4, 5, 8] of the CMM used single processor systems. We report here the implementation[9] of practical and accurate MD of massive (million atoms) systems on parallel processors. The current program is denoted as MPSim.

The next section of this article contains an overview of CMM and the methods used to calculate valence (bonded) interactions. The third section describes new results for large-scale simulations including rhinovirus and extrapolation of large argon clusters to the bulk limit. The implementation of CMM on parallel computers is given in the fourth section. The performance characteristics of the current code are presented in the fifth section. Conclusions are in the final section.

Cell Multipole Method

Because it is not feasible to explicitly calculate the interactions between all pairs of atoms in a large system, we must find a way to approximate the long-range region so as to decrease the costs to $O(N)$ while maintaining sufficient accuracy. A traditional approach is to ignore interactions beyond some cutoff. Unfortunately, this can lead to very large errors for Coulomb energies unless very long cutoffs are used.[6]

The key feature of multipole methods[4, 5, 7] is that long-range interactions are described using multipole fields representing large groups of atoms. This drastically reduces the number of computations, while including the effect of long-range terms. To maintain a given level of accuracy while minimizing cost, the nearby interactions are represented more accurately than the weaker, more distant interactions.

The long-range interaction portion of the simulation consists of computing and using these multipole expansions, as is the case for other multipole methods. The CMM is a particularly regular, easily parallelizable multipole method that enables this task to be performed efficiently. CMM consists of four parts:

1. *Octree decomposition*. The space occupied by the system of interest is divided into a hierarchical tree of cells so that long-range interactions can be treated using larger cells near the root of the tree, while short-range terms use smaller cells near the leaves. The effect of each cell is represented by a multipole expansion.
2. *Multipole moment expansion*. The multipole moments are computed for each cell at each level within the tree, starting from the leaves (smallest cells) and working toward the root of the tree (representing the entire system).
3. *Far-field multipole potential expansion*. The multipole moments from step 2 are used to describe the long-range fields due to the atoms within a given cell. However, to calculate the forces on any one atom within a cell we need to sum the fields due to all other distant cells in the system. These forces are described by expanding the fields for the distant cells in a Taylor series expansion about the center of the cell where the forces will be calculated. These terms are summed to obtain the total far field as a single Taylor series expansion. These Taylor series coefficients are computed from the multipole moments from step 2, starting at the root of the tree and working towards the leaves. In carrying out this expansion we exclude the effects of the atoms in neighbor cells since they will be included explicitly in the next step.

4. *Atomic forces*. Once the multipole potential expansions in step 3 are computed for each leaf cell, the force on each atom is computed as a combination of the far field evaluated at the atom location, the explicit nonbond interactions with nearby atoms, plus the short-range (valence) forces. These forces are then used in Newton's equation to update velocities and coordinates. Because the far field potential changes slowly with time only step 4 needs be carried out for each step of the MD.

OCTREE DECOMPOSITION

In CMM (as used for finite cases), the system of interest is surrounded by a *bounding box* (typically a cube, but in general a parallelepiped specified by three side lengths and three corner angles). This box, the level 0 cell, is then subdivided into eight octants by bisecting each side. The eight "child" cells are then further subdivided into octants to form grandchildren cells, which are in turn subdivided in octants, etc., forming an octree decomposition of the original box. The maximum level (M_{level}) of the tree is a parameter of the method and is chosen to obtain the best combination of speed and accuracy. Increasing M_{level} increases speed but at the cost of decreased accuracy and increased memory. The cells at this M_{level} are referred to as the "leaf" cells.

Unlike adaptive multipole methods,[10] in CMM the octree decomposition is carried out to the same level across the entire system. This is appropriate for applications to chemical, biological, and materials systems because the variations of density are seldom larger than an order of magnitude (cells with no atoms are ignored in the calculations). This regularity simplifies determining the neighbors of any given cell and increases the number of timesteps before it is necessary to rebuild the cell tree.

NONBOND INTERACTIONS

We consider nonbond interactions of the form:

$$E_{NB} = C_{unit} \sum_{i>j} \frac{q_i q_j}{R_{ij}^{\epsilon}} \tag{a}$$

where $\epsilon \geq 1$ is an integer and the other terms are defined in what follows. We use units of kcal/mol for E and Å for distances. Various special cases are:

1. *Coulomb interactions*. Here, $\epsilon = 1$, the q_i are particle charges in electron units, and:

$$C_{unit} = \frac{332.0637}{\varepsilon_0} \tag{b}$$

   where ε_0 is the static dielectric constant (we use $\varepsilon_0 = 1$ for a vacuum).
2. *Shielded Coulomb interactions*. For systems in which a polar solvent (particularly water) is treated implicitly, a common approach for Coulomb interactions is to use eq. (a) with $\epsilon = 2$.
3. *van der Waals dispersion*. The interaction constants can be written as:

$$E_{disp} = \sum_{i>j} \frac{C_{ij}}{R_{ij}^6} = \sum_{i>j} \frac{\sqrt{C_{ii}C_{jj}}}{R_{ij}^6} \tag{c}$$

   where the standard combination rule:

$$C_{ij} = \sqrt{C_{ii}C_{jj}} \tag{d}$$

   is assumed. In eq. (a) we use $\epsilon = 6$, $q_i = \sqrt{C_{ii}}$, and $C_{unit} = 1$.
4. *van der Waals repulsion (Pauli orthogonality)*. The repulsive short-range nonbond interactions are often described as:

$$E_{rep} = \sum_{i>j} \frac{A_{ij}}{R_{ij}^{12}} = \sum_{i>j} \frac{\sqrt{A_{ii}A_{jj}}}{R_{ij}^{12}} \tag{e}$$

   or:

$$E_{rep} = \sum_{i>j} A_{ij} e^{-B_{ij}R_{ij}} \tag{f}$$

   Using eq. (e) with eq. (c) leads to the Lennard–Jones 12-6 potential, whereas eq. (f) with eq. (c) leads to the exponential-6 or Buckingham potential. Term (e) can be treated using CMM. However, these terms [using (e) or (f)] fall off sufficiently fast with R that long-range calculations are not important. Thus, we treat case 4 using truncation, rather than multipoles.

MULTIPOLE EXPANSIONS

The potential far from a collection of charges can be written as the multipole expansion:

$$V(\mathbf{R}) = qR^{-\epsilon} + (\vec{\mu} \cdot \mathbf{R})R^{-\epsilon-2} + \tfrac{1}{2}(\mathbf{R} \cdot \mathbf{Q} \cdot \mathbf{R})R^{-\epsilon-4} + \cdots \tag{g}$$

where q, $\vec{u}$, $\mathbf{Q}$ describe the total charge, dipole moment, quadrupole moment, etc., for the collection of charges. The maximum level (M_{pole}) of multipole expansion to be used in eq. (g) is a parameter to be specified in CMM. Ding et al.[4,5] considered truncating at the quadrupole terms ($M_{pole} = 2$) as in eq. (g) or keeping the next level (or octopole, $M_{pole} = 3$) and found that $M_{pole} = 2$ leads to sufficient accuracy at reasonable costs. Increasing M_{pole} increases the accuracy at the cost of extra computation time.

The moments within a cell are calculated as:

$$\begin{aligned} q_{cell} &= \sum_{atoms} q_{atom} \\ \mu_{cell,\alpha} &= \sum_{atoms} \epsilon R_\alpha q_{atom} \\ Q_{cell,\alpha\beta} &= \sum_{atoms} \left[\epsilon(\epsilon+2)R_\alpha R_\beta - \epsilon R^2 \delta_{\alpha\beta}\right] q_{atom} \end{aligned} \tag{1}$$

where q, μ_α, and $Q_{\alpha\beta}$ are the charge, dipole, and quadrupole moments, respectively; $R_\alpha = R_{atom,\alpha} - R_{center,\alpha}$; $\vec{R}_{atom}$ is the position of the atom; $\vec{R}_{center}$ is the center of the expansion; the energy component is proportional to $R^{-\epsilon}$; and $\alpha = x, y, z$.

The centers of the multipole expansions can be chosen in several different ways. The simplest choice is to use the geometric centers of the cells. However, if the atoms within a cell are not distributed evenly, expanding about the centroid (or average) of the atom locations:

$$\vec{R}_{centroid} = \frac{1}{n} \sum \vec{R}_{atom} \tag{2}$$

(where n is the number of atoms in the cell) produces improved accuracy for a given level of multipole expansion. The centroids are computed hierarchically using the existing cell tree (the centroid of a higher level cell is equal to the weighted average of the centroids of its children, where the weights are the number of atoms in each child cell). The increased accuracy from the centroid formulation allows the use of more highly truncated expansions than would otherwise be required. The "Accuracy" subsection provides more details.

Charge-weighted and mass-weighted averages were tried, but these alternative centers were not found to significantly improve the accuracy.

Once the leaf cell multipole moments have been computed, the expansions may be translated to the centers of the next larger ("parent") cells and combined to obtain the multipole moments representing the parent cell:

$$\begin{aligned} q_{parent} &= \sum_{children} q_{child} \\ \mu_{parent,\alpha} &= \sum_{children} [\mu_{child,\alpha} + \epsilon R_\alpha q_{child}] \\ Q_{parent,\alpha\beta} &= \sum_{children} \Big[Q_{child,\alpha\beta} + (\epsilon+2) \\ &\quad \times (R_\alpha \mu_{child,\alpha} + R_\beta \mu_{child,\beta}) \\ &\quad + \epsilon(\epsilon+2) R_\alpha R_\beta q_{child} \\ &\quad - \left(2\vec{R}\cdot\vec{\mu} + \epsilon R^2 q_{child}\right)\delta_{\alpha\beta} \Big] \end{aligned} \tag{3}$$

This process continues through the tree until the root (level 0 cell) is reached. At this point, every cell in the system has associated with it a multipole expansion representing the field due to all of the atoms contained within the cell.

MULTIPOLE FIELD EXPANSIONS

To compute energies and forces on a given atom within the system, we need to obtain the field due to all of the other atoms in the system. This is broken into two components: the *near-field* interaction due to all of the atoms in the same cell or the 26 neighboring cells, and the *far-field* interaction due to all of the rest of the atoms. In CMM, the near-field interactions are evaluated explicitly to ensure high accuracy while the far-field interactions are evaluated using effective fields.

The far-field potentials to be applied to any atom in a given cell are combined into a Taylor series expansion about the center of the cell. This potential is then used to calculate the energy at any point within the cell. The coefficients of this expansion are determined from the multipole expansions computed in the previous subsection.

The Taylor series expansion of the farfield is written as[4]:

$$V(\vec{R}) = V_0 + \sum_\alpha V_\alpha R_\alpha + \sum_{\alpha\beta} V_{\alpha\beta} R_\alpha R_\beta + \cdots \tag{4}$$

where $\vec{R} = \vec{R}_{atom} - \vec{R}_{center}$ and V_0, V_α, and $V_{\alpha\beta}$ are the expansion coefficients. The centers used for the multipole field expansion are the same as those used for the multipole moment expansion.

Given the multipole potential (Taylor) expansion for a cell's parent, the remaining contributions needed to obtain a cell's far-field expansion are:

(i) the fields from the 8 children of each of the parent's 26 neighbors, plus

(ii) the fields from the other 7 children of the parent, minus

(iii) the fields from the cell itself and its 26 immediate neighbors at its level.

This leads to 27 cells (parent and its neighbors) times 8 (children per cell) minus 27 cells (cell and its neighbors), or 189 cells. All children of the same parent can be omitted because they are immediate neighbors of each other. Thus, to the parent's far-field expansion we add the fields from the 189 parent cell's neighbors' children (PNCs) to form the far field of the child (understanding that the other immediate neighbors of the cell in question are excluded).

To start this process, note that all cells at level 1 (the first set of eight child cells) are immediate neighbors of each other, and thus there is no far-field contribution from the parent or the PNCs of these cells.

By induction, we can continue generating Taylor expansions all the way to the leaf cells, at which point every cell has a Taylor expansion representing the far field within the cell.

We define the following useful terms:

$$\vec{R} = \vec{R}_{center,\,PNC} - \vec{R}_{center,\,cell} \tag{5}$$

$$\mu = \sum_{\alpha} \mu_{\alpha} R_{\alpha} \tag{6}$$

$$Q = \sum_{\alpha\beta} Q_{\alpha\beta} R_{\alpha} R_{\beta} \tag{7}$$

The contributions from each PNC to the expansion coefficients are then:

$$V_0 = R^{-\epsilon} q - R^{-\epsilon-2}\mu + \tfrac{1}{2} R^{-\epsilon-4} Q \tag{8}$$

$$\begin{aligned} V_{\alpha} = {} & \epsilon q R^{-\epsilon-2} R_{\alpha} - R^{-\epsilon-2}\left[(\epsilon+2) R^{-2} R_{\alpha}\mu - \mu_{\alpha}\right] \\ & + R^{-\epsilon-4}\left[\frac{1}{2}(\epsilon+4) R^{-2} R_{\alpha} Q - \sum_{\beta} Q_{\alpha\beta} R_{\beta}\right] \end{aligned} \tag{9}$$

$$\begin{aligned} V_{\alpha\alpha} = {} & \frac{1}{2}\epsilon q R^{-\epsilon-2}\left[(\epsilon+2) R^{-2} R_{\alpha}^2 - 1\right] \\ & + R^{-\epsilon-4}(\epsilon+2)\mu_{\alpha} R_{\alpha} \\ & - \frac{1}{2}(\epsilon+2)\mu R^{-\epsilon-4}\left[(\epsilon+4) R^{-2} R_{\alpha}^2 - 1\right] \\ & + \frac{1}{2} R^{-\epsilon-4} Q_{\alpha\alpha} \\ & + \frac{1}{4}(\epsilon+4) Q R^{-\epsilon-6}\left[(\epsilon+6) R^{-2} R_{\alpha}^2 - 1\right] \\ & - (\epsilon+4) R^{-\epsilon-6} R_{\alpha} \sum_{\beta} Q_{\alpha\beta} R_{\beta} \end{aligned} \tag{10}$$

$$\begin{aligned} V_{\alpha\beta} = {} & \epsilon(\epsilon+2) q R^{-\epsilon-4} R_{\alpha} R_{\beta} \\ & + (\epsilon+2) R^{-\epsilon-4} \\ & \times\left[\mu_{\alpha} R_{\beta} + \mu_{\beta} R_{\alpha} - (\epsilon+4) R^{-2}\mu R_{\alpha} R_{\beta}\right] \\ & + R^{-\epsilon-4} Q_{\alpha\beta} \\ & - (\epsilon+4) R^{-\epsilon-6} \sum_{\gamma}(Q_{\alpha\gamma} R_{\beta} R_{\gamma} + Q_{\gamma\beta} R_{\alpha} R_{\gamma}) \\ & + \frac{1}{2}(\epsilon+4)(\epsilon+6) Q R^{-\epsilon-8} R_{\alpha} R_{\beta} \end{aligned} \tag{11}$$

where the q, μ, and Q multipole expansion coefficients are those of the PNC.

The contributions from the parent cell's Taylor series coefficients to one of its child cells' coefficients are:

$$V_{child,0} = V_0 + \sum_{\alpha} V_{\alpha} R_{\alpha} + \sum_{\alpha\beta} V_{\alpha\beta} R_{\alpha} R_{\beta} \tag{12}$$

$$V_{child,\,\alpha} = V_{\alpha} + V_{\alpha\alpha} R_{\alpha} + \sum_{\beta} V_{\alpha\beta} R_{\beta} \tag{13}$$

$$V_{child,\,\alpha\beta} = V_{\alpha\beta} \tag{14}$$

where $\vec{R} = \vec{R}_{center,\,child} - \vec{R}_{center,\,parent}$ here, and the V coefficients on the right are from the parent cell.

FAR-FIELD EVALUATION AND NEAR-FIELD COMPUTATION

Once the cell Taylor expansions have been determined, computing the interaction energy and force due to the far-field at each atom is simply a matter of evaluating the Taylor series at the atom's position [given by eq. (4)] and calculating the interaction of the field and the atomic charge.

Because the far field changes much more slowly than the near field, it is feasible to perform the far-field calculation at intervals of every 5 to 10 timesteps. The centers and coefficients of the Taylor series expansions representing the far field are kept constant during the interval.

The remaining near-field interactions between an atom and the other atoms in the same cell and in neighboring cells are computed explicitly, using the appropriate charge-charge interaction equations (Coulomb or van der Waals).

VALENCE FORCE FIELD CODE

The valence force field (FF) consists of interactions defined in terms of the bonds and angles between atoms in a molecule. These terms are used to parameterize the quantum-mechanical effects due to stretching the chemical bonds and bending the angles.

Among the terms we use are:

(a) Bond stretch: harmonic or Morse potentials:

$$E_b = \tfrac{1}{2}K_b(R - R_e)^2 \tag{15a}$$

or:

$$E_b = D_e[e^{-\alpha(R-R_e)} - 1]^2 \tag{15b}$$

where R_e is the equilibrium bond length, K_b is the force constant, D_e is the bond energy, and $\alpha = \sqrt{K_b/2D_e}$ is the Morse scaling parameter.

(b) Angle bend: harmonic or harmonic-cosine potentials:

$$E_a = \tfrac{1}{2}K_\theta(\theta - \theta_e)^2 \tag{16a}$$

or

$$E_a = \tfrac{1}{2}C_\theta(\cos\theta - \cos\theta_e)^2 \tag{16b}$$

where θ_e is the equilibrium angle, K_θ the force constant, and $C_\theta = K_\theta/\sin^2\theta_e$.

(c) Torsion: cosine expansions:

$$E_\phi = V_0 + \frac{1}{2}\sum_{n=1}^{12} V_n * \cos(n\phi) \tag{17}$$

where V_n is the barrier for the nth-fold rotation.

(d) Inversion: harmonic-cosine potentials:

$$E_\psi = \tfrac{1}{2}K_\psi(\cos\psi - \cos\psi_e)^2 \tag{18}$$

where ψ_e is the equilibrium angle and K_ψ the force constant.

(e) Cross terms: combinations of the above terms used to improve the accuracy of vibrational frequencies: For two atoms I and K bonded to J, we have:

$$E_{bond-angle} = K_{r1\theta}(\cos\theta - \cos\theta_e)(R_1 - R_1^e) + K_{r2\theta}(\cos\theta - \cos\theta_e)(R_2 - R_2^e) \tag{19a}$$

$$E_{bond-bond} = K_{rr}(R_1 - R_1^e)(R_2 - R_2^e) \tag{19b}$$

where $R_1 = R_{IJ}$ and $R_2 = R_{JK}$ and the Ks are force constants. For atoms in the dihedral I—J—K—L, we include:

$$E_{angle-angle} = K_{aa}f_x(\phi)(\cos\theta_{IJK} - \cos\theta_{IJK}^e) \times (\cos\theta_{JKL} - \cos\theta_{JKL}^e) \tag{19c}$$

to couple the angle I—J—K with the angle J—K—L. The $f_x(\phi)$ factor depends on the dihedral $\phi_{I-J-K-L}$ and has the form $f_7(\phi) = (1 - 2\cos\phi)/3$, $f_1(\phi) = \cos\phi$, or $f_0(\phi) = 1$.

Important features of the above valence FF terms are:

1. A variety of functional forms is associated with each type of interaction to allow an accurate description of the FF.
2. The functions depend on the coordinates of two, three, or four atoms. The input file only provides a list of bonds between atoms (equivalent to edges between nodes in a graph). The program deduces the three- and four-body interactions from the two-body connectivities.
3. The functional forms and the constants in each function depend on the types of the atoms participating in each interaction. These types typically involve the element or atomic number of the atom plus information about hybridization or oxidation state of the atom.

All of these points must be taken into account to produce a useful, general MD code, but they also make parallelization of the valence FF computation more difficult. We therefore describe some of the techniques used to overcome this problem.

Each term is assigned to exactly one of the atoms participating in the interaction. To minimize the distance between the responsible atom and the other atoms in the interaction, the central atom of each angle term and the center atom of each inversion term is designated as the responsible atom. For torsions, one of the two middle atoms is used.

For each term for which an atom is responsible, the type of term (angle/torsion/inversion), the numbers of the other atoms participating in the interaction, and an index into an array of structures for that term type defining the functional form and constants for the interaction are stored in a list associated with that atom. Functions are provided to iterate through all interactions of a given type.

Cross-terms are not stored as separate interactions; rather, they modify the functional forms of the relevant three- or four-body interactions.

Bonds are treated slightly differently, as their information is also needed to maintain the molecular connectivity. For each atom, the other atoms bonded to it and the index to the array of bond function/constant structures are stored in a separate list associated with that atom.

These lists are computed only once because the bonding network does not change.

Each CPU can compute all the interactions for the atoms residing on that CPU provided that it has access to the atomic coordinates for each atom participating in any such interaction. In physical systems, all atoms participating in an interaction must be close in space, and we assume that all atoms participating in any given valence interaction are either in the same leaf cell or in a neighboring leaf cell. With this assumption all atomic coordinates for all atoms of interest to the valence interactions will have already been communicated to the cell during the near-field computation of the CMM. They are merely saved until the valence interactions have been computed. On machines that provide a shared memory programming model, this assumption need not be embedded in the code, but it still serves to increase performance by ensuring that needed data will be in local memory rather than on a remote node.

Each computation produces a force vector for each atom that must be added to all other force components for that atom. For the computed partial forces to be added into each atom's force vector, we accumulate these forces locally on each node both for the local atoms assigned to the cell and for the nonlocal atoms in neighboring leaf cells that were involved with a valence term assigned to the cell. Then, after all the valence FF calculations have been completed, we send partial forces of nonlocal atoms back to the node from which the atomic coordinates were obtained. That node then adds the incoming partial forces into the local atom force vectors.

In the current implementation, each atom participating in a bond interaction is responsible for computing half of the energy contribution and the partial force exerted on itself by the bond. Thus, partial bond forces need not be communicated. Given that partial forces for the other terms are being communicated, this inefficiency will be eliminated in the future.

On machines with a message-passing architecture, each node creates a list of all remote cells required to perform its own calculations. It then sends a request to the appropriate nodes asking for that remote cell/atom information.

As the calculation progresses, the force components on remote atoms are updated. When these updates are complete, each CPU goes through its list of remote atom data, bundles the updated information, and sends it back to the CPU owning the remote atoms. Each CPU then receives updates made to its local atoms and adjusts the force vectors appropriately. Subsequent calculations involving an atom's force vectors then proceed.

Three other terms related to the valence FF are computed rather differently:

1. *Nonbond exclusions*. Because the valence FF usually implicitly includes any contributions from nonbonded interactions between atoms that are bonded to each other (1–2 interactions) or bonded to a common atom (1–3 interactions), the corresponding nonbonded energies and forces included in the CMM must be excluded from the final result. (In addition, for the AMBER FF,[11] the 1–4 terms are also scaled, requiring similar corrections.) This can be implemented in two ways:

   (i) avoid the computation of these nonbonded interactions during the inner loop of the near-field calculation in CMM; or
   (ii) subtract out their effects as a separate step.

   Method (i) reduces floating-point operations, but at the cost of many memory references to do table look-ups. Method (ii) computes some interactions twice and increases errors due to subtraction of large numbers, but it avoids conditional branches and uncached memory accesses in performance-critical code. We use method (ii) in the current implementation.

2. *Hydrogen bonds*. For the nonbond interaction between a hydrogen atom and a hydrogen bond acceptor, some FFs use a modified functional form in place of the usual 6–12 vdW term. For example, the AMBER FF[11] uses:

$$E_{ij}(R) = AR^{-12} - BR^{-10} \qquad (20)$$

   where R is the distance between the acceptor atom (N, O, F, S, Cl) and the hydrogen atom of the donor. These are implemented as a modified nonbond exclusion, in which the original energy and force is subtracted while the new term is added.

3. *Off-diagonal nonbonds*. Explicit in the use of CMM is the geometric mean combination rule in eq. (d), which is used in eq. (c) and (f). This can be written as follows:

$$(D_0)_{ij} = \sqrt{(D_0)_{ii} \cdot (D_0)_{jj}} \tag{21a}$$

$$(R_0)_{ij} = \sqrt{(R_0)_{ii} \cdot (R_0)_{jj}} \tag{21b}$$

Some FFs prefer the use of an arithmetic mean:

$$(D_0)_{ij} = \sqrt{(D_0)_{ii} \cdot (D_0)_{jj}} \tag{22a}$$

$$(R_0)_{ij} = \tfrac{1}{2}\left[(R_0)_{ii} + (R_0)_{jj}\right] \tag{22b}$$

Sometimes (particularly for hydrogen bonds) it is necessary to use a special, different functional form to express the van der Waals component of the nonbond energy between two types of atoms.

Our implementation allows the $(D_0)_{ij}$ and $(R_0)_{ij}$ to be different than in eq. (d) or eq. (21) for specific pairs of atoms.

As for the valence FF terms, the atoms participating in nonbond exclusions are determined at initialization. Each atom has a list of other atoms for which the exclusions should be computed.

Off-diagonal nonbonds (and their special case, AMBER-type hydrogen bonds) cannot be predefined by an unchanging list. Instead, we generate this list each time the CMM far-field is updated. We again assume that only interactions with atoms in nearest-neighbor leaf cells are of interest. All pairs of atoms within a leaf cell or in adjacent leaf cells are tested to see if they have both a hydrogen-bond donor and acceptor or have atom types for which an off-diagonal nonbond has been defined. Those pairs that qualify are then added to the list.

At each timestep, the pairs of atoms in the exclusion lists have the CMM-calculated interaction subtracted. For off-diagonal nonbond interactions, a new energy term is then calculated and added.

All of the above terms are sufficiently general to implement the functional forms and constants used in standard molecular mechanics force-fields such as AMBER[11] and DREIDING II.[12]

INTEGRATION

In the current implementation, the dynamics step in the simulation uses either a simple Verlet integrator (for microcanonical dynamics)[13] or a Nosé–Hoover constant-temperature integrator (for canonical dynamics).[14] In either case, the atomic forces are integrated to generate the new atomic velocities, which are in turn integrated to give new atomic positions. The equations of motion are as follows:

Verlet:

$$\vec{v}_{n+\frac{1}{2}} = \vec{v}_{n-\frac{1}{2}} + \frac{\vec{f}_n}{m}\Delta t \tag{23}$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_{n+\frac{1}{2}}\Delta t \tag{24}$$

Nosé–Hoover:

$$\vec{v}_n = \frac{1}{1 + \zeta_n \frac{\Delta t}{2}}\left(\vec{v}_{n-\frac{1}{2}} + \frac{\vec{f}_n}{m}\frac{\Delta t}{2}\right) \tag{25}$$

$$\vec{v}_{n+\frac{1}{2}} = \vec{v}_n\left(1 - \zeta_n\frac{\Delta t}{2}\right) + \frac{\vec{f}_n}{m}\frac{\Delta t}{2} \tag{26}$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_{n+\frac{1}{2}}\Delta t \tag{27}$$

$$\sigma_n = \sigma_{n-\frac{1}{2}} + \zeta_n\frac{\Delta t}{2} \tag{28}$$

$$\sigma_{n+\frac{1}{2}} = \sigma_n + \zeta_n\frac{\Delta t}{2} \tag{29}$$

$$\zeta_{n+1} = \zeta_n + \frac{1}{\tau_s^2}\left(\frac{T_{n+\frac{1}{2}}}{T_{bath}} - 1\right)\Delta t \tag{30}$$

$$KE_{bath} = \frac{3N}{2}k_B T_{bath}\tau_s^2\zeta_n^2 \tag{31}$$

$$PE_{bath} = (3N + 1)k_B T_{bath}\sigma_n \tag{32}$$

$$T_n = \sum_i \frac{m_i v_{i,n}^2}{(3Nk_B)} = \frac{KE_n}{(\frac{3}{2}Nk)}$$

$\vec{f}$, $\vec{v}$, $\vec{x}$, and m are the atomic forces, velocities, coordinates, and mass, respectively. The subscripts n, $n + \frac{1}{2}$, and $n + 1$ denote increasing timesteps. ζ is the Nosé–Hoover friction variable responsible for transferring heat from the system to the bath or vice versa, σ is its integral, and τ_s is the time constant associated with the heat transfer. The instantaneous kinetic energy, $KE_{n+\frac{1}{2}}$, is written in terms of a temperature, $T_{n+\frac{1}{2}}$; this must be evaluated at the half-step because it is used to compute $\dot{\zeta}_{n+\frac{1}{2}}$, which is needed to obtain ζ_{n+1}. T_{bath} is the (constant) bath temperature. N is the number of atoms in the system.

The MPSim program also allows for periodic systems (RCMM).[5] The applications here discuss only the finite case.

Applications

ARGON CLUSTER

We previously studied[15] the properties of small clusters as a function of size and developed an approach for extracting quantities which could be used to predict the free energy or arbitrarily large clusters and for the infinite crystal. These calculations were based on MacKay clusters with up to 561 atoms.

It is well known from both experiment and theory that van der Waals complexes (Ar,[16] CH_4,[17] etc.) lead to icosahedral clusters with particular stabilities for certain *magic numbers* corresponding to Mackay icosahedral structures,[18] whereas the bulk crystal leads to a face-centered cubic (fcc) structure. For Ar, the Mackay icosahedral structure is most stable for up to about 1600 atoms, the bulk fcc structure is most stable above about 10^5 atoms, and other structures are stable in the intermediate size range.[19] The magic number Mackay icosahedral structures are composed of concentric filled shells surrounding a central atom.

Using MPSim we have now explicitly calculated structures for very large Mackay clusters (up to 10^7 atoms) and for large fcc clusters. These results are used to scale the properties to infinite systems. These are among the largest simulations of Lennard–Jones (LJ) systems. Lomdahl and coworkers[20] were the first to study multimillion-particle systems. They simulated a two-million LJ particle system at 0 K in two dimensions as it failed under one-dimensional strain.[20]

We use a simple LJ 12-6 potential with equilibrium separation $R_e = 3.82198$ Å and well depth $D_e = 0.23725$ kcal/mol. These were determined to fit the structure ($a = 5.31$ Å) and cohesive energy ($E_{coh} = 1.848$ kcal/mol) of Ar crystal at 0 K.

To test the accuracy of MPSim, we quenched a Mackay icosahedral cluster and a spherical fcc cluster, both with 10,179 atoms. With a cell occupancy of 10,179 (i.e., with all atoms in the same cell), we calculated a binding energy of $80219.2 * D_e$ for Mackay and $80243.3 * D_e$ for FCC. These results agree, to all six significant figures, with the literature values.[21]

The accuracy of the method as a function of average occupany and CMM level is depicted in Figure 1. Figure 1d shows that to obtain a structure with an rms coordinate error of 0.05 Å requires an accuracy in the rms force error of 0.01 (kcal/mol)/Å. Figure 1c illustrates that, for 100,000 atoms this accuracy is obtained with level 5 which Figure 1a shows to have about 10 particles per cell.

FCC Structures

Because the optimal structure for large clusters is likely fcc or nearly fcc, we chose to quench magic number spherical fcc clusters to determine their maximum binding energy.

Table I summarizes the minimization parameters, maximum binding energy, and rms force for each of the clusters studied. The fcc clusters have binding energies per particle (E/n) which are consistent with known values for smaller clusters and the bulk structure. To estimate the limits for an infinite crystal we plot (E/n) versus $n^{-1/3}$ in Figure 2. We found that going beyond $n^{-4/3}$ in the polynomial expansion leads to insignificant improvement in the rms error. Thus we truncated the expansion at the fifth-order term.

$$\frac{PE}{\mu D_e} = a + bn^{-1/3} + cn^{-2/3} + dn^{-1} + en^{-4/3} \tag{33}$$

The fitting parameters are $a = -8.60990 \pm 0.00041$, $b = +15.74416 \pm 0.037$, $c = -11.89436 \pm 0.77$, $d = +17.19378 \pm 1.1$, and $e = -20.38740 \pm 1.5$. This fit is somewhat less precise than that of Xie et al.[19] when applied to small clusters, but is more accurate for extrapolation to large clusters. We have determined the binding energy contribution to the chemical potential, surface energy, and Tolman length (see Fig. 2). As discussed elsewhere:[15]

- **(i)** the intercept a gives the bulk chemical potential, μ;
- **(ii)** the slope b at $n^{-1/3} = 0$ is proportional to the surface tension, σ; and
- **(iii)** the ratio c/b of the curvature to the slope at $n^{-1/3} = 0$ is proportional to the Tolman length, δ.

We use our previous estimate[15] for the zero-point energy and limit our discussion to 0 K to highlight the current results without extraneous complications.

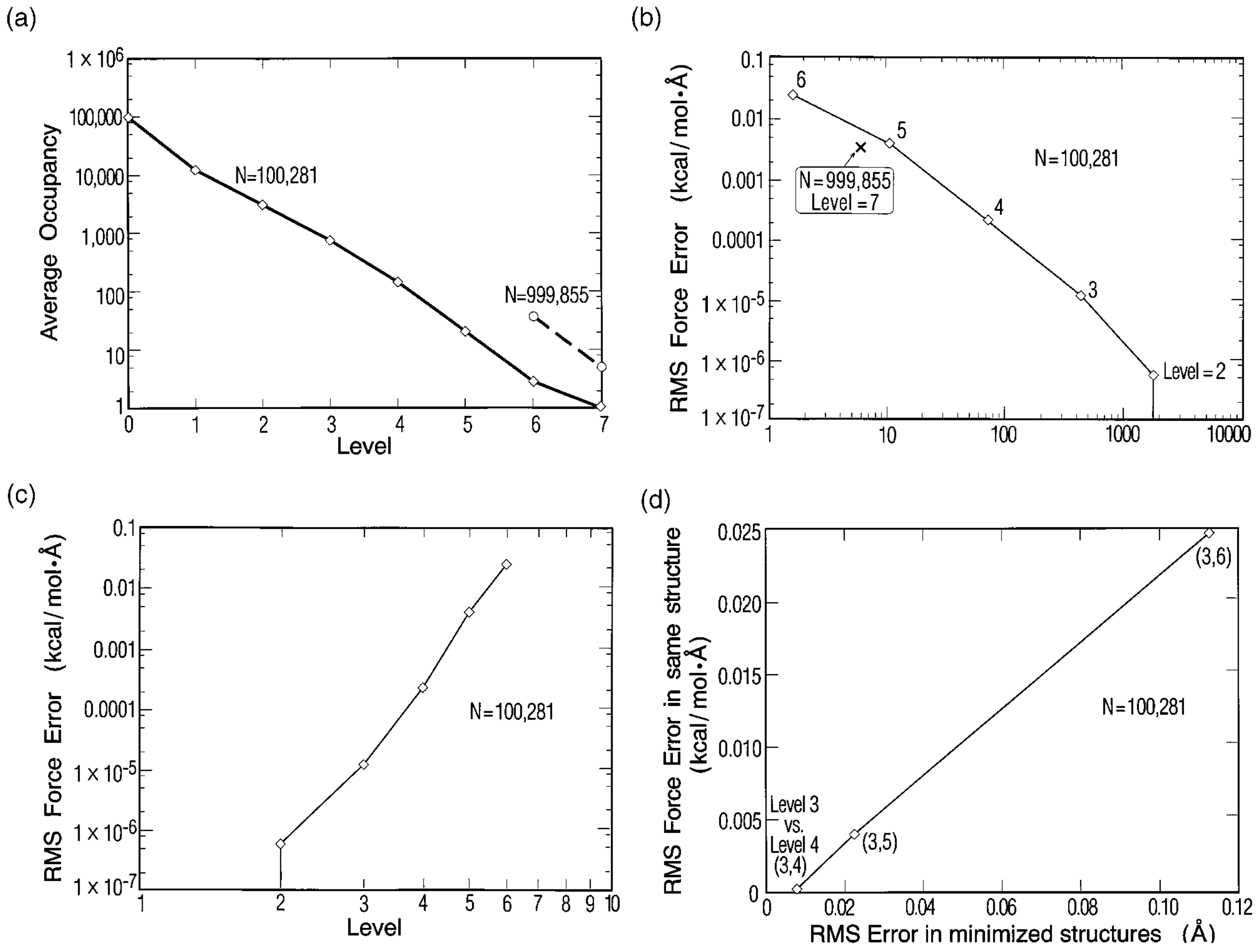

FIGURE 1. Dependence of the accuracy of CMM on the octet level for $(Ar)_{100,000}$. Empty cells are not counted in computing occupancy; thus the lowest possible occupancy is one. A cube of side = 240 Å was placed around the $(Ar)_{100,000}$ cluster (diameter = 190 Å). (a) The average occupancy vs. level. (b) The rms force error vs. average occupancy. (c) The rms force error vs. level. (d) The rms force error vs. error in minimized structure.

The parameters from eq. (33), at $n^{-1/3} = 0$, together with our previously published zero-point energy estimate,[15] allow us to estimate the chemical potential (μ), surface tension (σ), and Tolman length (δ) at 0 K.

$$\mu = D_e \times a + 30.861 h\nu_{char} \quad (34)$$

$$\sigma = (D_e \times b - 34.413 h\nu_{char}) \big/ [(4\pi)^{1/3}(3v)^{2/3}] \quad (35)$$

$$\delta = \frac{1}{2}\left(-D_e \times \frac{c}{b} + 10.594 h\nu_{char}\right)\left(\frac{3v}{4\pi}\right)^{1/3} \quad (36)$$

Here, v is the molecular volume at 0 K and $h\nu_{char}$ is a characteristic zero-point energy.[15] We expect these estimates to be more accurate than previous efforts because all three properties are dominated by the binding energy contribution (as opposed to the zero-point energy contribution) and our estimates benefit from minimizations of very large clusters (see Table I and Fig. 2). This illustrates one use of massive molecular dynamics in probing the asymptotic approach of cluster properties toward the bulk limit.

Using the values for argon ($v = 37.456$ Å^3/atom, $h\nu_{char} = 6.181$ cal/mol) gives the following estimates:

$$\mu_{Ar}^{0\,K} = -1.852 \text{ kcal/mol}$$

$$\sigma_{Ar}^{0\,K} = 45.2 \text{ dyn/cm}$$

$$\delta_{Ar}^{0\,K} = 0.81 \text{ Å}$$

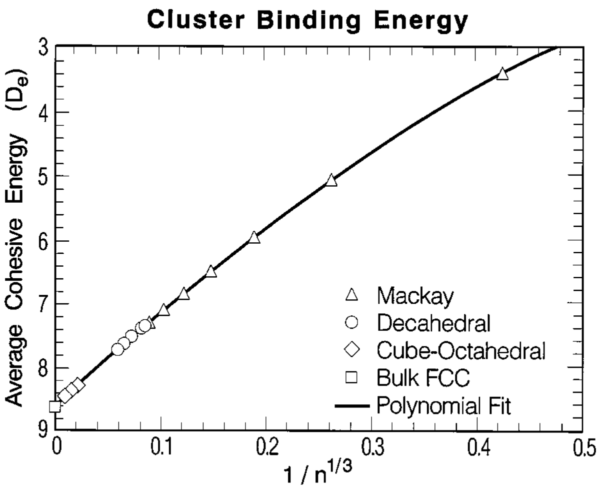

FIGURE 2. Argon binding energy per particle (in units of D_e) versus cluster size. Diamonds are Mackay icosahedral structures,[19] triangles are decahedral structures,[21] circles are spherical fcc structures (current work), and the square is the bulk fcc structure. The intercept, slope, and curvature at the origin are related to the binding energy contribution to the chemical potential, surface tension, and Tolman length, respectively.

The surface tension and Tolman length for cryogenic solids are difficult to measure experimentally. We know of no prior experimental determinations with which to compare these data.

MOLECULAR DYNAMICS SIMULATIONS ON RHINOVIRUS 14

Rhinovirus is the major cause of common cold in humans.[1] It belongs to the picornavirus family of animal viruses which consist of an icosahedral protein coat wrapped around the RNA.[1] The viral coat of the rhinovirus is made up of 60 icosahedral units, each of which consists of four proteins: VP1, VP2, VP3, and VP4. Of these, VP1, VP2, and VP3 form the surface of the coat, while VP4 is buried inside close to the RNA. The external diameter of the virion is about 300 Å.

One unit of the protein coat, as simulated, consists of 8546 atoms (including H and counterions) of which 6924 are seen in X-ray diffraction studies. The RNA structure is not determined by the X-ray.

Acidification of the viral coat is necessary for viral uncoating which ejects the RNA from the core of the virion.[22,23] For polio and rhinoviruses this acidification step is postulated to be required for an infection to proceed normally (although there is some debate about this[23b]). It is also known that the rhinovirus structure becomes disordered on acidification near the ion binding site on the icosahedral fivefold axis.

The X-ray structure of the rhinovirus 14 has been solved[1] to a resolution of 3.0 Å. It is important to note that the X-ray structure for this virus is derived under the *assumption* of icosahedral symmetry. Although this symmetry may be globally correct, it cannot be correct at the atomic level for sites like the pentamer vertex (fivefold axis) which would require a penta-coordinated metal. Thus, enforcing symmetry in the structure leads to errors or unresolved atoms at such sites. Because these sites may be important in describing infection by the virus, it is useful to carry out MD simulations on the whole virus without assuming symmetry. This is the only way to obtain structural information about such sites.

We used MD simulations to observe the structural changes that occur during acidification.[22] An

TABLE I.
Properties for Minimized Argon Clusters. Here PE Is the Total Cohesive Energy and rms Is the Root-Mean-Square Force (Indicates Convergence of Minimization).

Particles	Structure[a]	Levels	Average occupancy	PE $\left[\frac{\text{kcal}}{\text{mol}}\right]$	$(PE/D_e)/n$ $\left[\frac{\text{kcal}}{\text{mol} \times \text{particle}}\right]$	rms force $\left[\frac{\text{kcal}}{\text{mol} \times \text{Å}}\right]$
10,179	co-fcc	0	10,179	19,032.0	7.88085	
10,179	Mackay	0	10,179	19,037.7	7.88322	
100,281	sph-fcc	3	737	196,858	8.2743	0.0027
299,677	sph-fcc	3	295	595,622	8.3775	0.0012
999,855	sph-fcc	5	153	2,005,405	8.4540	0.00061
∞	fcc	—	—	—	8.6102	0

[a]co-fcc = cube-octahedral fcc; Mackay icosahedral; sph-fcc = spherical fcc; fcc = bulk fcc.

understanding of these structural changes should be useful in designing new antiviral agents to inhibit uncoating of the viral coat of the rhinovirus. These simulations were carried out using MPSim. Rhinovirus 14 (Fig. 3) requires both valence and nonbond forces in the simulation.

To allow for hydrogen bonding, hydrogens were added where appropriate to oxygen, nitrogen, and sulfur atoms of the X-ray structure of the asymmetric unit. Due to the acidic and basic residues there was a net charge of −5 on each asymmetric unit. In water, such exposed atoms lead to solvation by water dipoles and free ions near the charge. In our calculations we simulated this solvation by adding counterions (Na^+ and Cl^-) close to the side chains of the acidic (Asp, Glu) and basic residues (Lys, Arg). Not every charged residue requires a counterion, because many are involved in salt bridges. We eliminated such paired side chains by examining all pairwise distances between the central atoms of oppositely charge side chains. These atoms are C_δ in aspartic acid, C_δ in glutamic acid, C_δ in arginine, and N_δ in lysine. Pairs less than 10 Å apart were considered to stabilize one another, and were not given countercharges, since these pairs may be involved in salt bridges. This automated procedure of adding counterions resulted in adding 1500 counterions to the whole virus (900 Na^+ and 600 Cl^-).

The X-ray structure is unresolved for the first 28 residues of the VP4 protein which is buried inside close to the RNA. The RNA is also not included in

FIGURE 3. The minimized structure of rhinovirus 14 starting from the X-ray structure 4RHV in the Brookhaven database. The shades of red show the pentameric site formed by the VP1 subunit. VP2 is in green (or shades of green) and VP3 is in blue (or shades of blue). VP4 atoms are buried inside and some of them are seen in yellow.

the current calculations, because no structure is available for it (it does not have icosahedral symmetry and hence is invisible in the X-ray).

The asymmetric unit thus has 8546 atoms which includes the heteroatom hydrogens, the counterions placed to neutralize the charges, and the crystallographic waters. The full viral coat of rhinovirus 14 has 512,760 atoms. Explicit waters in the calculation would improve the accuracy but were excluded. This omission is compensated partially in two ways:

1. By using a distance-dependent dielectric to mimic the charge-shielding capacity of water between charges i and j:

$$E(q_i, q_j) = \frac{q_i q_j}{\varepsilon_{eff} r_{ij}} = \frac{q_i q_j}{\varepsilon r_{ij}^2} \tag{36}$$

2. By using counterions to balance charges on the side chains of residues.

The whole virus structure, along with the crystallographic waters and the counterions, was minimized using the AMBER FF.[11] A steepest-descent method (dynamics at 0 K) was used. The timestep was set to 0.75 fs since the lone pairs on the sulfurs are included explicitly. The FF parameters used for the counterions Na^+ and Cl^- are those corresponding to the hydrated ions.[24] Hydrated ions would attenuate the electrostatic interaction between the ions and the charged groups in the side chains. The rms force on the final minimized structure is 0.2 kcal/mol/Å. The radius of gyration of the starting structure was 133.49 Å and remains unchanged (0.001%) during the minimization. The rms difference in coordinates between the starting structure and the minimized structure is 0.268 Å. This value was calculated considering only the atoms in the viral coat (excluding the crystallographic waters and the counterions).

The minimized structure was then equilibrated at 300 K using Nosé–Hoover (constant temperature) dynamics. The Nosé time constant was chosen to be $\tau_{nose} = 7.5$ fs which is ten times[25] the timestep used for integration. The MD was done for 15 ps at 300 K until equilibrium was reached. The temperature of the system reached an equilibrium after 5 ps of dynamics to an average value of 295.99 K. We observe that the equilibrated virus structure does not disassemble in spite of the RNA being absent. Further studies on the effect of lowering the pH are ongoing.

Implementation

We implemented the CMM for message-passing parallel multicomputers. Unlike some other parallel multipole method implementations,[26] this is a three-dimensional, fully distributed code, allowing the solution of large-scale problems that would not fit in the memory of a single processor.

The pseudocode for the algorithm used is presented in Appendix A of the Supplementary Material (see footnote on p. 501; also available at http://www.wag.caltech.gov).

MESSAGE TYPES

Five types of messages form the heart of the CMM algorithm:

1. Cell center sent from a child cell to its parent cell.
2. Multipole moments sent from a child cell to its parent cell.
3. Multiple moments sent from a cell to its PNCs.
4. Multipole far-field (Taylor series coefficients) sent from a parent cell to its children.
5. Atoms sent from a leaf cell to its neighbors.

The first four are used to compute the far-field and occur only every N_{update} timesteps; the last type of message must be executed at every timestep.

The cell centers need to be sent through the tree before the multiples because each parent cell must determine the location of its center before processing incoming multipoles from its children.

Additional messages are used for initialization, synchronization, flow control, and atom reassignment.

ACTIVE MESSAGES

An active message model was used for the development of the code. In this model, reception of a message triggers the execution of a function specified in the message with the message contents passed as an argument or arguments to the function.[27] This model allows low-latency communications by avoiding copying. It also leads to a natural, asynchronous, multithreaded style of programming. We believe that the advantages of the active message model will make it the preferred programming model for future generations of multicomputers.

Currently, the active message model is the preferred model for programming experimental fine grain parallel processing hardware such as the MIT J- and M-machines,[28] which are among the targeted platforms for this code.

As an example, the second type of message from the previous section, involving the communication of multipoles from a cell to its parent, is implemented through an active message of the type CHILD, with the destination CPU, destination (parent) cell, and sending cell's multipoles as arguments. When such a message is received on the destination CPU, it triggers the invocation of the `child()` routine given in the pseudocode (Appendix A) which parses the arguments, obtains the appropriate destination cell, and calls a purely computational routine to combine the child's multipoles with the parent's.

Although active messages have not been extensively used on traditional multicomputers such as Intel's Delta and Paragon,[29] we have found that refinements can be added to improve performance on such architectures, primarily by gathering together multiple small messages into a few large messages. This buffering significantly reduces per-byte overhead.

Because the fundamental messaging primitives have no flow control, we implemented a flow control system on top of the active messages. Each processor is allowed to send a certain number of messages (a window) to each other processor in the system. When the window is full, it must wait until it has received acknowledgments before sending more messages. Keeping the windows on a per-processor basis allows more messages to be outstanding (not acknowledged) than if a single window were used on each processor. This in turn reduces the amount of nonproductive busy-waiting on each processor.

The acknowledgment can often be packed with message data needed to continue the calculation. In addition, we could optimize further by sending buffered messages taking into account the size of the system message buffer, indicated during program invocation. This strategy not only handles flow control on the underlying message system, but reduces the latency costs by sending fewer larger buffers, rather than more smaller buffers.

There are two possible messaging strategies: a ''pull'' strategy in which data is requested from another processor, and a ''push'' strategy in which data is sent to a destination processor. The ''pull'' strategy requires two messages per data transfer, while the ''push'' strategy could possibly be optimized to only use one message per transfer, if useful data can always be attached to flow-control acknowledgments. The expected savings and optimization flexibility led us to use the latter strategy.

DATA DISTRIBUTION

In CMM, communications occur across the surface area of the set of cells assigned to each CPU; minimizing this surface area while distributing the data is therefore highly desirable. To accomplish this, we assign cells to CPUs using ranges of cell indices designed to keep the surface area low.

The index for each cell is based on the octree; a cell's number is equal to its parent's number multiplied by 8 plus an index varying from 0 to 7. This ensures that consecutively numbered cells are generally close to each other in space. In particular, any range of cell numbers tends to form one or two approximately-cubic domains. Although this partitioning may not be ideal, it works well, even on highly irregular (noncubic) systems and is simple to implement.

A cell's number can thus be represented by a sequence of octal (base 8) digits, each corresponding to the child index at a different level. Within this numbering system, the numbers of a cell's parent and children can be computed using simple expressions:

$$n_{child} = 8n_{parent} + index_{child} \tag{37}$$

$$n_{parent} = \left\lfloor \frac{n_{child}}{8} \right\rfloor \tag{38}$$

The number of a cell's neighbor in, say, the $-x$ direction is determined by a slightly more complex set of operations as given in the following C code:

```
mask = 01111111111; /* octal */
nmask = ~ mask;
x = cell & mask;
if ( x>0) {
          return (x − 1) & mask |
          (cell & nmask);
}
else {
         return −1; /* no such
         neighbor */
}
```

The CMM involves tree-based computations and computations involving pairs of cells at the same level of the tree. The former are inherently ordered, either from the leaves to the root or vice versa, and so cannot deadlock.

The PNC computations are the first kind of same-level calculation. These interactions use multipole information from PNCs to generate Taylor series expansion coefficients in a cell. Because there are no dependencies on the result during this step, deadlock is again impossible.

The other kind of same-level calculation involves the computation of interactions between pairs of atoms. Deadlock may be avoided in one of two ways: either by having the interaction computed by both the processors containing the atoms involved, or by having the interaction computed by only the processor containing the lower-numbered atom. In the first case, there is again no dependence on the output; in the second, any output dependence is resolved by the ordering of the atom numbers. In actual implementation, it is usually easier to substitute entire cells and their numbers for atoms in the above description, but the effect is identical.

LOAD BALANCING

To a first approximation, the computational time per timestep is dominated by the near-field interactions, which in turn are dependent on the number of atoms (or occupied cells) assigned to each processor. Therefore, we arrange for each processor to be responsible for a consecutively numbered range of cells containing no less than n_{atoms}/n_{cpus} atoms (except for the last CPU, which contains all remaining atoms). The use of ranges keeps the surface area low, as mentioned above, and also limits the size of the tables needed to determine on which CPU a given cell resides to merely one integer per CPU.

Because atoms may move between cells and thus may occasionally move between processors during the MD simulation, it may become necessary to rebalance the load. Each CPU can determine how many atoms it contains; a simple, rapid linear sweep through the CPUs then would readjust the cell ranges and communicate the cell and atom data to their new locations.

More sophisticated dynamic load balancing algorithms can also be implemented within this scheme. The version of the code for the KSR architecture uses global synchronization timing information, collected across user-specified intervals, to readjust the cell ranges to achieve near-perfect load balancing. A processor with excess work will not need to wait for global synchronizations, while one with too little work will need to wait for those with too much. The algorithm determines the accumulated waiting time on each processor, calculates the average across all processors, and then moves cells (by adjusting cell ranges) from those processors with below-average waiting times to those with above-average waiting times. This process dynamically adjusts the load balance during the calculation to fit the observed behavior of the code on the machine, allowing it to compensate for irregularities in the simulated system or for other user or system tasks running on the processors.

Levels in the tree closer to the root are assigned in such a way as to generally minimize the amount of communication required during tree traversals. A parent cell is assigned to the same CPU as its zeroth numbered child. Given the load-balancing-determined ranges of leaf cells on each CPU, it is simple to determine the CPU containing a higher level cell by a simple shift and binary search.

INPUT / OUTPUT

Input data to the simulation includes a control file giving parameters for the dynamics, a FF file giving parameters for the various terms in the energy expression, and a set of structure files, one per CPU, containing the atomic position, charge, and connectivity information. These structure files are generated by a load balancing preprocessor from a single, unified structure file based on the number of CPUs to be employed. The input data set totals 80–120 bytes per atom (depending on the system's connectivity).

Because large systems of atoms can span hundreds of megabytes per input file per processor, we found this approach to be stressful for the relatively small number of I/O nodes handling the disks on the Intel Paragon and Delta. When each processor opens its own file, the read requests funnel through very few I/O nodes (16 on the 512 compute node Paragon on which we ran our timings). The I/O nodes become overloaded retrieving information spread across the disks, so we must throttle our requests. Another approach would have been to reorganize the set of input structure files into a single, unified file such that each processor could seek to its own place in the file and read. This would have allowed us to take advantage of the Intel-optimized global read/write calls, but moved away from the single reader/

writer mode which the J-machine prefers. Having all the processors read all the atoms from the structure file and discarding those that are not local is considered too expensive for the multigigabyte data sets of very large systems.

Output includes the potential and kinetic energy at each timestep, as well as other parameters of the dynamics. At user-specified intervals, snapshots of the system are taken containing positions, velocities, and forces on each atom. These allow analysis of the properties of the system (including evolution with time) and also serve the important purpose of allowing the simulation to be restarted. The output files contain 72 bytes per atom, plus a small header.

Long production runs require the flexibility of restarting (due to machine crashes and varying scheduling policies) sometimes with a different number of processors than the original run. Because the startup and checkpoint files are written on a per-processor basis, a change in the number of processors requires reassignment of cells and atoms (and load rebalancing). With the current I/O architecture this was infeasible (a unified input file might make this easier).

Although the calculation per timestep might be longer on fewer processors, the available CPU time for small to medium numbers of processors is often much greater than similar blocks of time for large numbers of processors. Thus, time to solution for large systems benefits from having the ability to run on various-sized partitions.

SHARED MEMORY IMPLEMENTATION

The KSR AllCache architecture[30] presents a shared memory programming model to the user, even though it is implemented on top of a physically distributed memory. This model greatly simplifies the initial parallel programming task and, when attention is paid to the actual location of data, can provide high parallel efficiency without extensive rewriting of code. Two key features of the KSR architecture are used in the code. First, all data is stored in a single global address space. This means that data not specified as private is automatically sharable by all processors, merely by accessing a global variable or dereferencing a pointer. Second, any 128-byte ‘‘subpage’’ (the basic quantum of memory sharing in the machine) may be locked automatically by a processor so that only it retains access rights to the subpage. Any other processor attempting to lock (or even reference) the subpage must wait until the locking processor explicitly releases it. Though an exclusive lock primitive is often provided in parallel programming systems, the KSR version is substantially more powerful because it is associated with 128 bytes of data. In particular, the cost of a lock is small in terms of execution time and zero in terms of memory usage (provided that the data being locked is at least 128 bytes). These values are significantly cheaper than most implementations provide. The caching strategy of the machine allows data to be passed from CPU to CPU merely by referencing it. As long as the new CPU continues to use a piece of data, it remains in the cache for this CPU. The copy of the data in the old CPU will be flushed as necessary or when the data is modified. Because of the global address space, no bookkeeping is needed in the software to keep track of where a given piece of data is. This contrasts with message-passing architectures, in which data must explicitly be sent from one CPU to another, and it is often necessary to maintain elaborate structures to identify the location of a desired piece of data.

Since the KSR architecture benefits less from parallel I/O, the per-CPU structure and other files described in the previous section are instead combined into one file, read by one CPU and distributed via the shared memory to the other CPUs as required. Though this increases the startup time for the code, it also increases the code’s flexibility, as any job may be run or restarted on any number of CPUs.

Performance

ACCURACY

The accuracy of the CMM is quite good, even at the level of quadrupole expansions. Coulombic charges are much smaller than masses in gravitational N-body simulations, resulting in less error due to truncation of the expansions. Although the ‘‘charges’’ used for the R^{-6} London force calculation are larger, the interaction strength falls off much more rapidly with distance, again allowing earlier truncation of the expansions than others have found practicable for nonchemical systems.[31]

A test was run on a 12,431-atom argon cluster. The potential energy of the system with all pairwise nonbond interactions included was -2.14966×10^4 kcal/mol. Using the CMM on this finite system with a bounding cube 180 Å on a side, subdividing to a maximum level of 5 (so that the average number of atoms per occupied leaf cell, κ, was 4.0 atoms) gave a potential energy of -2.14479×10^4 kcal/mol, a difference of only 0.2%.

This error is well within the margins commonly accepted for molecular mechanics simulations. In particular, it is much smaller than the typical error when using spline cutoff nonbonds, in which only interactions with a given radius are considered, with a spline smoothing function applied to remove the discontinuity in the energy function at the boundary. Even with only van der Waals forces, using a spline cutoff with inner radius 8.0 Å and outer radius 8.5 Å gives a potential energy error of 6.2% (-2.01544×10^4 kcal/mol).

The CMM-derived forces are in error by about the same amount as the spline cutoff forces. The maximum force error from CMM was 5.70×10^{-2} kcal/mol/Å, with the overall rms error being a mere 1.57×10^{-2} kcal/mol/Å. Spline gave a similar maximum error of 3.97×10^{-2} kcal/mol/Å and an overall rms error of 1.91×10^{-2} kcal/mol/Å.

Since the maximum force error is several times the overall rms error, most errors are in fact small, with only a few particles having inaccurate force vectors.

Despite the improved accuracy of CMM over spline, CMM costs less CPU time. With spline cutoffs the above calculation took three times longer to perform than with CMM on a single CPU (KSR-1, 20 MHz). The time for spline was: 98.9 seconds (82.9-second setup and 16.0 seconds required for every timestep), versus CMM: 31.7 seconds (23.1-second setup and 8.6 seconds required for every timestep).

Using cell centers instead of cell centroids reduces the accuracy of the method somewhat. The potential energy increases to -2.14059×10^4 kcal/mol, an error of 0.4%, or double the previous error, but still far less than spline cutoff. The maximum force error increases to 7.17×10^{-2} kcal/mol/Å, and the overall rms error increases to 2.07×10^{-2} kcal/mol/Å.

SCALABILITY

Each of the steps comprising the CMM is linear and scalable, or nearly so. There are seven steps in the CMM; these may be divided into two major parts. The five steps of the first part compute the far field (the multipole potential Taylor series expansions representing the field from atoms far away from each atom), while the two steps of the second part compute the near field (the explicit calculation of effects due to atoms near each atom).

The first step, generation of the leaf cell multipoles, is fully linear and runs in parallel because there are no data dependencies.

The second and third steps, computation of the cell centers and propagation of the multipoles to the root of the tree, both require a traversal of the octree. Since the number of cells in the system is the sum of a geometric series with a logarithmic number of terms, it is proportional to the number of atoms:

$$n_{leaves} = \frac{n_{atoms}}{\kappa} \tag{39}$$

$$n_{levels} = \log_8 n_{leaves} \tag{40}$$

$$n_{cells} = \sum_{i=0}^{n_{levels}} 8^i = \frac{8 n_{atoms}}{7\kappa} - \frac{1}{7} \tag{41}$$

where κ is the number of atoms per cell at the finest level.

This analysis assumes that the number of leaves is linearly dependent on the number of atoms in the system. We subdivide until the average number of atoms per occupied leaf is about 4, so this assumption obviously holds for systems with uniform density.

Real systems have density fluctuations at a large scale, although at the fine scale of a few atoms, there tend to be regions of constant density interspersed with voids. To handle these void regions efficiently in the calculations, we ignore any cells that are unoccupied, allocating them no memory or computation time. The number of occupied leaves thus remains the same as for a uniformly dense system. Additional levels, and thus additional higher level cells, may be required for such systems, but these additional cells will be relatively few in number for all but pathological systems.

Each pass through the tree involves a constant number of computations per cell and therefore is linear in the number of atoms in the system. The tree traversals cannot be made fully parallel, however, as there are increased data dependencies near the root of the tree. Because the number of computations to be done near the root is relatively small, due to the high degree (8) of the octree, the tree traversal time is dominated by the computations near the leaves, which are highly parallel.

The fourth and fifth steps, the PNC computation, and the propagation of the Taylor series coefficients to the leaves of the tree are also linear and highly parallel (as argued above).

The two steps of the near-field computation, neighbor-cell interactions, and within-cell interactions, are perfectly linear and also execute in parallel, limited only by the communications overhead of transmitting atoms from leaf cells to their neighbors.

Note that since the long-range interactions have been sped-up by using the multipole approach, computing these near-field interactions (the short-range forces) now becomes the dominant cost of the overall computation, especially when the multipoles are not updated at every dynamics step ($N_{update} > 1$).

Finally, the integration step has only one data dependency, a global sum to determine the overall kinetic energy, with the rest being perfectly parallel and linear.

There are various synchronization steps in the algorithm; the current (suboptimal) implementation of these scales approximately linearly in n_{cpus}.

The total amount of computation that occurs is thus linear in the number of atoms. Nonlinearities in the scaling with number of CPUs are primarily the result of load-balancing inefficiencies.

The toal amount of communication required is almost linear in the number of atoms, except for the tree effects just described. The fraction of this communication occurring off-node depends on the number of CPUs being used. Generally, it will vary as the total surface area of the boundaries between cells assigned to each CPU. Because the number of cells is proportional to the number of atoms, the number of cells assigned to each CPU is proportional to $n_{atoms} n_{cpus}^{-1}$. The surface area of this set of cells is proportional to the 2/3 power of this number, or $n_{atoms}^{2/3} n_{cpus}^{-2/3}$. Multiplying this by the total number of CPUs gives a communication cost proportional to approximately $n_{atoms}^{2/3} n_{cpus}^{1/3}$.

Further complicating the analysis, however, is the fact that much of this communication can itself occur in parallel. The amount of communication can also be decreased by taking into account the fact that an atom or PNC may need to interact with multiple cells on the same destination node. This avoidance of redundant transmissions has been implemented for the PNC multipole communication step, but not yet for the near-field atom communication step.

The best case time is thus:

$$t_{timestep} = C + t_{comp} \frac{n_{atoms}}{n_{cpus}} + t_{comm} \qquad (42)$$

while the worst case time is:

$$t_{timestep} = C + t_{comp} \frac{n_{atoms}}{n_{cpus}} (1 + k_{loadbal}) + t_{comm} n_{atoms}^{2/3} n_{cpus}^{1/3} \qquad (43)$$

where C is constant setup overhead, t_{comp} is the computation time per atom, t_{comm} is the communication time per cell, and $k_{loadbal}$ represents the overhead due to imperfect load balancing.

The program was tested for performance on a series of multimillion-atom argon cluster systems. Although these systems do not include Coulombic charges and their interactions, all Coulomb terms were still calculated (and correctly resulted in zero energy and zero force), and hence are included in the timing results. The calculations were run on the Intel Paragon XP/S of the CSCC using OSF/1 Release 1.0.4. Five cluster sizes were used: 1 million, 2 million, 5 million, 8 million, and 10 million atoms.

For a constant number of atoms, the plot of the logarithm of time versus the logarithm of the number of CPUs will lead to a line of slope -1. As n_{cpus} becomes large, the slope should level off, eventually increasing to a value of 1/3 (assuming that imperfect load balancing depends little on the number of CPUs used).

Figure 4 shows such a graph of log(time) against log(CPUs) for the far-field Taylor series generation process. The number of CPUs along the x-axis ranges from 64 to 512. Three lines are drawn to show the scaling for systems of different sizes ranging from 1 million to 5 million atoms. The 8-

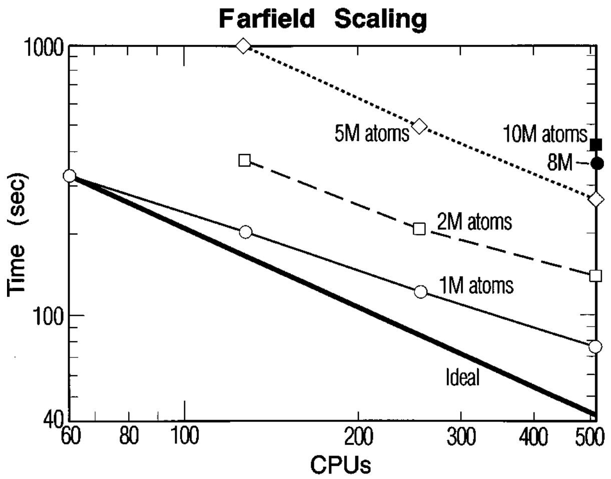

FIGURE 4. Scaling of computational cost with number of Intel Paragon CPUs for the far-field computation.

and 10-million atom systems could only be run on all 512 CPUs. The thick line shows the slope that would be achieved for ideal (perfectly linear) scaling.

This portion of the calculation contains all of the tree manipulations. The effects of the data dependencies inherent in the tree (which cause imperfect parallelization) can be seen by the less negative slope of the lines, especially for the smallest (1-million-atom) system. Larger systems, in which the amount of computation per node increases, show better scaling, which is more nearly parallel to the ideal line. We would only reach the regime of zero or positive slope in pathological cases with much too little computation for the amount of communication required (i.e., too few atoms spread across too many CPUs).

The far-field computation is only performed at intervals of N_{update} timesteps, so its imperfect scaling has a relatively small effect on the overall time to solution.

Figure 5 shows the same type of graph, but for the near-field and integration computations. This portion of the calculation contains no tree-derived data dependencies and its scaling curves are close to parallel with the ideal line.

For a constant number of CPUs, the plot of the time per atom versus the number of atoms ideally should give a constant. Deviation from a constant line should be most apparent at small numbers of atoms, because the deviation is expected to scale as $n_{atoms}^{-1/3}$. These graphs are shown in Figure 6 for the far-field computation and Figure 7 for the near-field and integration steps (the x-axis is the number of atoms in the simulated system in millions and the y-axis is the time spent in the indicated portion of the code divided by the number of atoms).

We see that the lines of 128, 256, and 512 CPUs are close to flat, with the expected upturn for the far-field computation at small system sizes. To show that this is in fact due to the communications overhead (as described in the theoretical scaling formula above), we plotted the time per atom against the number of atoms to the $-1/3$ power.

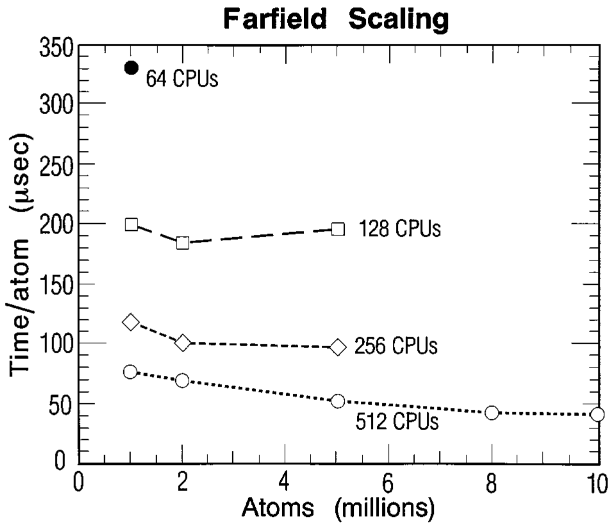

FIGURE 6. Scaling of computational cost with the number of atoms for far-field computation.

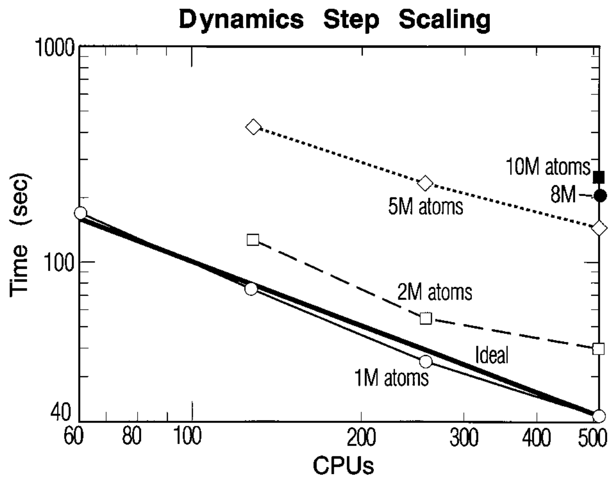

FIGURE 5. Scaling of computational cost with number of Intel Paragon CPUs for the near-field and integration steps.

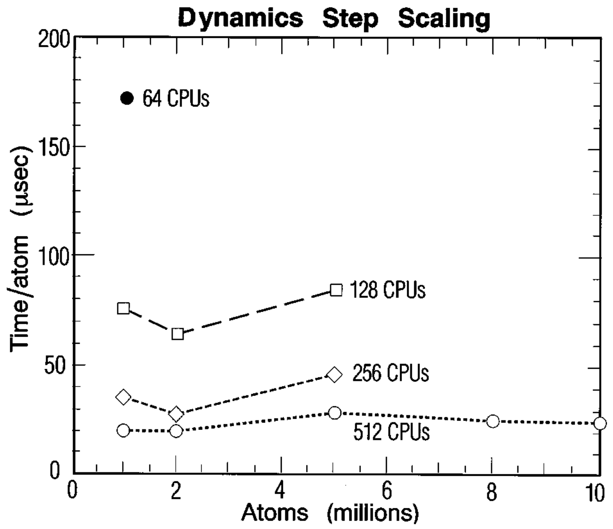

FIGURE 7. Scaling of computational cost with the number of atoms for near-field and integration steps.

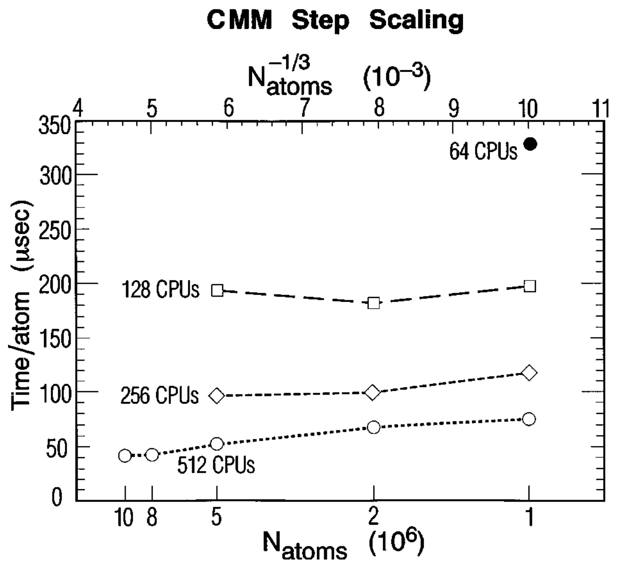

FIGURE 8. Scaling of computational cost with size ($N_{atoms}^{-1/3}$) for the CMM step.

This should give lines with a slope of zero in the best case, or $t_{comm} n_{cpus}^{1/3}$ (a positive constant) in the worst case. In Figure 8, the lines of zero or constant positive slope show that the imperfect scaling for the far-field computation is in fact due to the communications term in the theoretical scaling formula.

We also demonstrated the scalability of the algorithm used for the valence computations with respect to the number of processors and with respect to the size of the system using the shared-memory architecture version of the code on a KSR-1. The rhinovirus coat was simulated, either alone, or as a dimer of about 1 million atoms.

On the KSR, we get close to linear scaling with both CPUs and atoms. We compute an expected value of t_{comp} from the time per timestep, the number of CPUs, and the number of atoms. Doing so gives the approximately equal values in Table II for the viral system, including valence force-field interactions. For comparison, the message-passing code gives t_{comp} values ranging from 5.6 to 18 CPU/ms/atom for the far-field step and 12 to 18 CPU/ms/atom for the near-field and dynamics step.

TABLE II.
Timing for Rhinovirus Calculations on the 64-Node KSR1.

Atoms (millions)	0.5	1.0	1.0	0.5	1.0
CPUs	30	30	45	60	60
Time (CPU ms / atom)	3.07	3.55	3.35	4.39	3.98

Conclusions

The MPSim code is efficient for simulating large-scale (millions of atoms) systems on massively parallel computers, using standard force fields including bonded interactions. The CMM is used to calculate the long-range forces efficiently, and calculation of the short-range forces has also been parallelized. The code has been demonstrated to perform accurate MD calculations on systems including large argon clusters and rhinovirus 14.

In the argon case, a formula for the potential energy of large clusters was fitted, and the chemical potential, surface tension, and Tolman length at 0 K were determined.

For rhinovirus 14, a minimized protein coat structure was generated using no symmetry assumptions, starting from the X-ray structure. This minimized structure was shown to be stable with respect to disassembly, even in the absence of the RNA contents, at neutral pH. This result reproduces the experimental observation that acidification is required for ejection of the RNA.

The code runs on both message passing and distributed shared memory architectures and has been shown to scale well as the number of processors is increased and as the number of atoms is increased well into the millions.

Acknowledgments

The Intel Paragon XP/S-L38 used for the production and timing runs in this study was provided courtesy of the Concurrent Supercomputing Consortium (CSCC). The Paragon has 512 compute processors, each using an Intel i860 XP microprocessor and containing 32 megabytes of memory. The system software used was Paragon OSF/1 Release 1.0.4 Server 1.2 R1.2. The Intel Touchstone Delta used for the production runs was also provided by the CSCC. Each of the Delta's 512 compute processors has an Intel i860 XR with 16 Megabytes of memory per processor. The computational portions of the code were developed on the Materials and Process Simulation Center's KSR-1/64, funded by an NSF Grand Challenge Application Grant (ASC 92-17368). Funding for this work was provided by DARPA and NSF GCAG (ASC 92-100368). Other support was provided by NSF (CHE 95-17368) and DOE (OIT/BCTR).

References

1. M. G. Rossmann, et al., *Nature*, **317**, 145 (1985).
2. D. A. Tomalia, A. M. Naylor, and W. A. Goddard III, *Angew. Chem.*, **29**, 138 (1990).
3. W. A. Goddard III, N. Karasawa, S. Dasgupta, R. Donnelly, J. Wendel, C. B. Muisgrave. H.-Q. Ding, K.-T. Lim, J. J. Gerdy, J.-M. Langlois, X. Chen, R. P. Muller, M. N. Ringnalda, R. Friesner, T. Maekawa, G. Miller, T. Yamasaki, T. Cagin, and A. Jain, In *Molecular Modeling; The Chemistry of the 21st Century*, M. A. Chaer Nascimento, Ed., World Scientific Publishing, Rio de Janeiro, Brazil, 1993.
4. H.-Q. Ding, N. Karasawa, and W. A. Goddard III, *J. Chem. Phys.*, **97**, 4309 (1992).
5. H.-Q. Ding, N. Karasawa, and W. A. Goddard III, *Chem. Phys. Lett.*, **196**, 6 (1992).
6. H.-Q. Ding, N. Karasawa, and W. A. Goddard III, *Chem. Phys. Lett.*, **193**, 197 (1992).
7. (a) A. W. Appel, *SIAM J. Sci. Stat. Comput.*, **6**, 85 (1985); (b) J. Barnes and P. Hut, *Nature*, **324**, 446 (1986); (c) L. Greengard and V. Rokhlin, *J. Comput. Phys.*, **73**, 325 (1987).
8. A. M. Mathiowetz, N. Karasawa, A. Jain, and W. A. Goddard III, *Proteins*, **20**, 227 (1994); see also A. M. Mathiowetz, Ph.D. thesis, Department of Chemistry, California Institute of Technology, Oct. 1992.
9. K.-T. Lim, Ph.D. thesis, California Institute of Technology, May 1995.
10. J. Carrier, L. Greengard, and V. Rokhlin, *SIAM J. Sci. Stat. Comput.*, **9**, 669 (1989).
11. S. J. Weiner, P. A. Kollman, D. A. Case, U. C. Singh, C. Ghio, G. Alagona, S. Profeta Jr., and P. Weiner, *J. Am. Chem. Soc.*, **106**, 765 (1984).
12. S. L. Mayo, B. D. Olafson, and W. A. Goddard III, *J. Phys. Chem.*, **194**, 8897 (1990).
13. (a) L. Verlet, *Phys. Rev.*, **159**, 98 (1967); (b) A. Rahman, *Phy. Rev.*, **136**, A405 (1964).
14. (a) S. Nosé, *Mol. Phys.*, **52**, 255 (1984); (b) S. Nosé, *J. Chem. Phys.*, **81**, 511 (1984); (c) W. G. Hoover, A. J. C. Ladd, and B. Moran, *Phys. Rev. Lett.*, **48**, 1818 (1982); (d) W. G. Hoover, In *Molecular Dynamics; Lecture Notes in Physics*, **258**, Springer, New York, 1986.
15. R. B. McClurg, R. C. Flagan, and W. A. Goddard III, *J. Chem. Phys.*, **102**, 3322 (1995).
16. J. A. Harris, R. S. Kidwell, and J. A. Northby, *Phys. Rev. Lett.*, **53**, 2390 (1984).
17. O. Echt, O. Kandler, T. Leisner, W. Miehle, and E. Recknagel, *J. Chem. Soc. Faraday Trans.*, **86**, 2411 (1990).
18. A. L. Mackay, *Acta Cryst*, **15**, 916 (1962).
19. B. Raoult, J. Farges, M. F. DeFeraudy, and G. Torchet, *Phil. Mag. B*, **60**, 881 (1989).
20. P. S. Lomdahl, D. M. Beazley, P. Tomayo, and N. Gronbech-Jensen, *Int. J. Modern Phys.*, **C4**, 1075 (1993).
21. J. Xie, J. A. Northby, D. L. Freeman, and J. D. Doll, *J. Chem. Phys.*, **91**, 612 (1989).
22. V. L. Giranda, B. A. Heinz, M. A. Oliveira, I. Minor, K. H. Kim, P. R. Kolatkar, M. G. Rossmann, and R. R. Rueckert, *Proc. Natl. Acad. Sci.*, **89**, 10213 (1992).
23. (a) R. R. Rueckert, In *Fundamental Virology, 2nd Ed., Picornaviridae and Their Replication*, B. N. Fields, Ed., Raven Press, New York, p. 357; (b) M. Gromier and K. Wetz, *J. Virol.*, **64**, 3590 (1990).
24. U. C. Singh, S. J. Weiner, and P. A. Kollman, *Proc. Natl. Acad. Sci. USA*, **82**, 755 (1985).
25. N. Vaidehi, A. Jain, A., and W. A. Goddard III, Constant temperature constrained molecular dynamics: the Newton–Euler inverse mass operator method, *J. Chem. Phys.* (to be submitted).
26. (a) J. A. Board, J. W. Causey, J. F. Leathrum, A. Windemuth, and W. Schulten, *Chem. Phys. Lett.*, **198**, 89 (1992); (b) L. Greengard and W. D. Gropp, *Comput. Math. Appl.*, **20**, 63 (1990).
27. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.
28. (a) W. J. Dally et al., The J-machine: a fine-grain concurrent computer, In *Information Processing 89*, G. X. Ritter, Ed., Elsevier, Amsterdam, 1989; (b) W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Fyler, *IEEE Micro.*, **12**, 23 (1992); (c) W. J. Dally et al., *M-Machine Architecture v1.0*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Concurrent VLSI Architecture Memo 52, Feb. 1994; (d) D. Masket, *A Message-Driven Programming System for Fine-Grain Multicomputers*, Masters thesis, California Institute of Technology, Feb. 1994; (e) D. Maskit et al., *System Tools for the J-Machine*, California Institute of Technology, Department of Computer Science Technical Report, CS-TR-93-12, 1993.
29. (a) URL: http://www.ssd.intel.com/homepage.html; (b) URL: http://www.ccsf.caltech.edu/annrep94/faciL_1.html.
30. In *KSR/Series Principles of Operation*, Kendall Square Research, Revision 7.0, March 15, 1994.
31. (a) J. K. Salmon and M. S. Warren, *Int. J. Supercomput.*, **8**, 129 (1994); (b) J. K. Salmon and M. S. Warren, *J. Comput. Phys.*, **111**, 136 (1994).